LA-UR-21-24774
Approved for public release; distribution is unlimited.

Title: LUNA Condition-Based Monitoring Update: Mahalanobis Distance for Excess Load and External Leak Cases

Author(s): Green, Andre Walter

Intended for: Progress report to sponsor.

Issued: 2021-05-17

LUNA Condition-Based Monitoring Update:

Mahalanobis Distance for Excess Load and External Leak Cases

Presented 5/11/2021

Andre' Green

Mahalanobis Distance

Mahalanobis distance is the distance between a point and a distribution.
The distribution is represented by several sample points, which are stacked into a matrix.

mahalanobis_distance (point, matrix) =
    (point – mean(matrix)) * inv(cov(matrix)) * (point – mean(matrix))^T

'point' is an N-element vector.
'matrix' is an [M x N] matrix.

'mean(x)' returns the column-means of a matrix x.
'inv(x)' returns x's inverse matrix.
'cov(x)' returns x's covariance matrix.

Mahalanobis Classifier

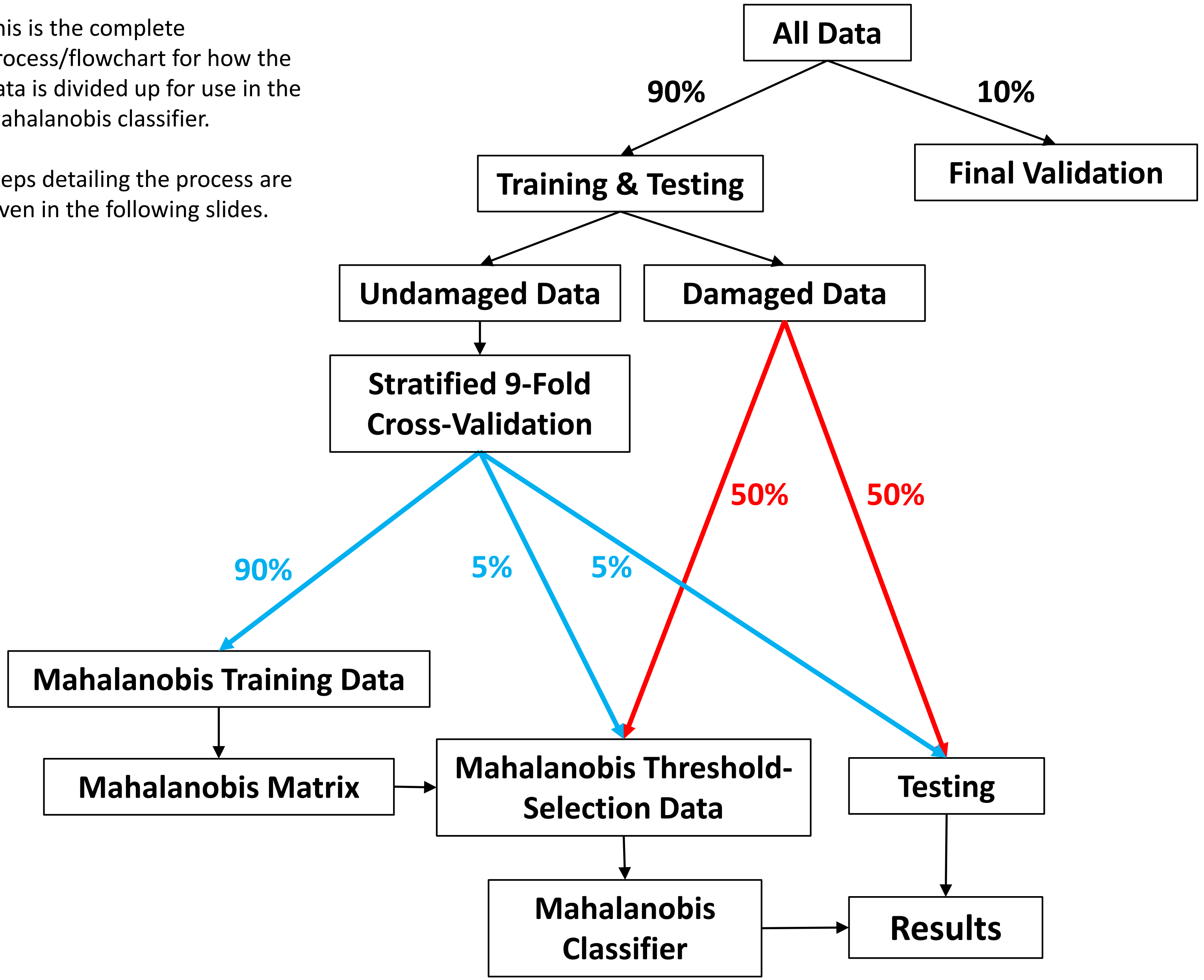
A binary Mahalanobis classifier is straightforward to make, if you know how to calculate the mahalanobis distance.

mahalanobis_classify(point, matrix, threshold) =
    mahalanobis_distance(point, matrix) > threshold

Mahalanobis Classifier

This is the complete process/flowchart for how the data is divided up for use in the mahalanobis classifier.

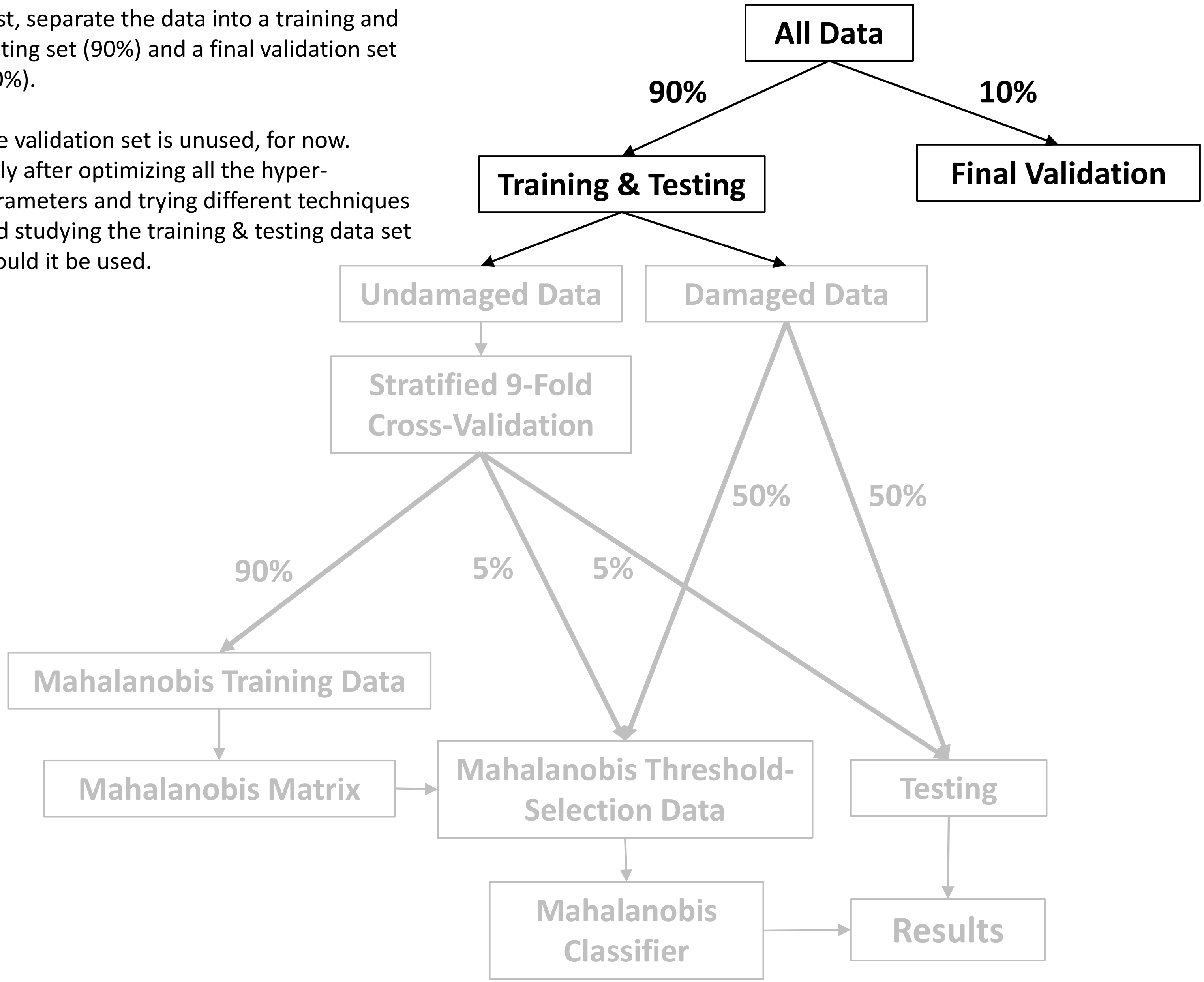
Steps detailing the process are given in the following slides.



Mahalanobis Classifier

First, separate the data into a training and testing set (90%) and a final validation set (10%).

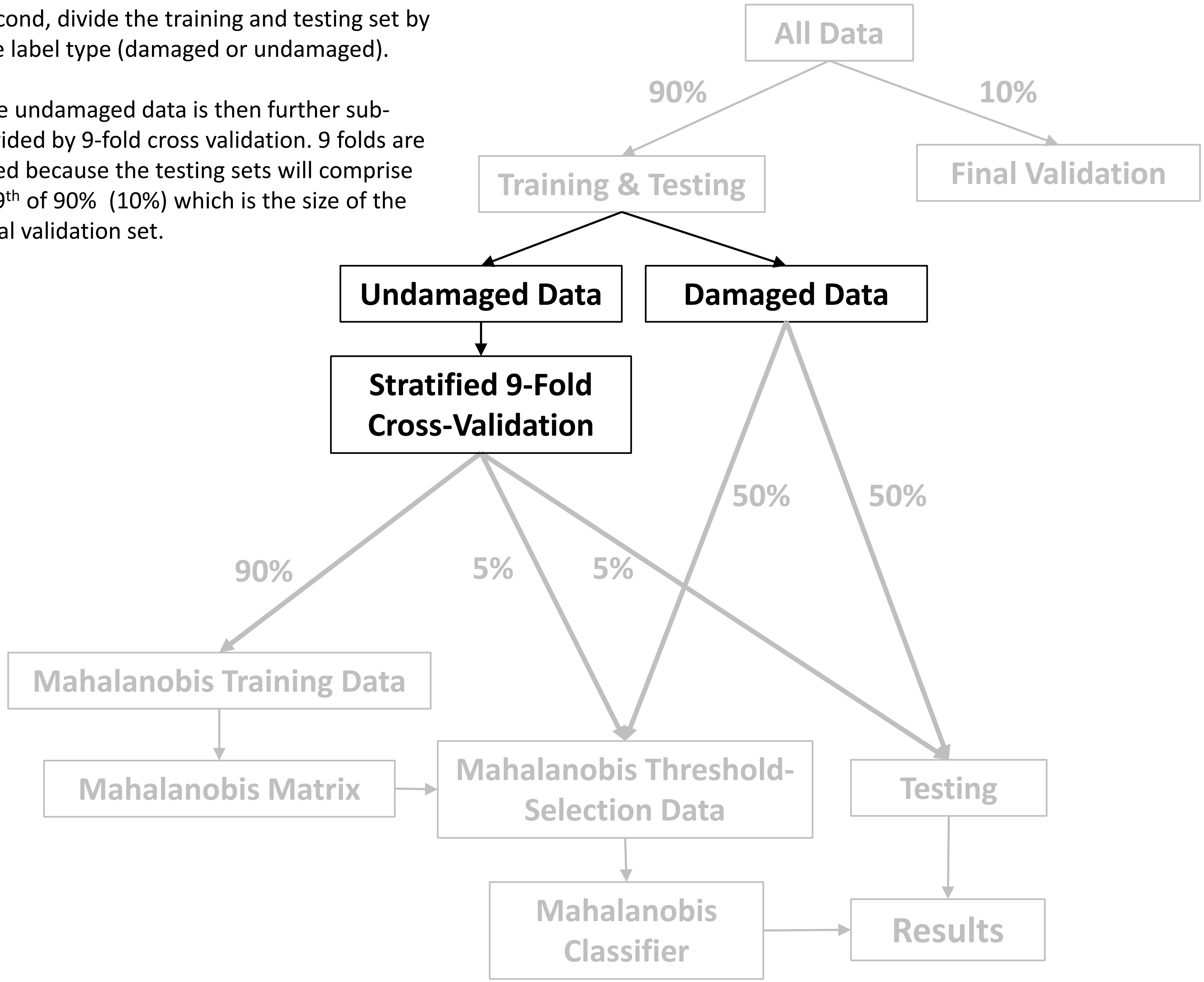
The validation set is unused, for now. Only after optimizing all the hyper-parameters and trying different techniques and studying the training & testing data set should it be used.



Mahalanobis Classifier

Second, divide the training and testing set by the label type (damaged or undamaged).

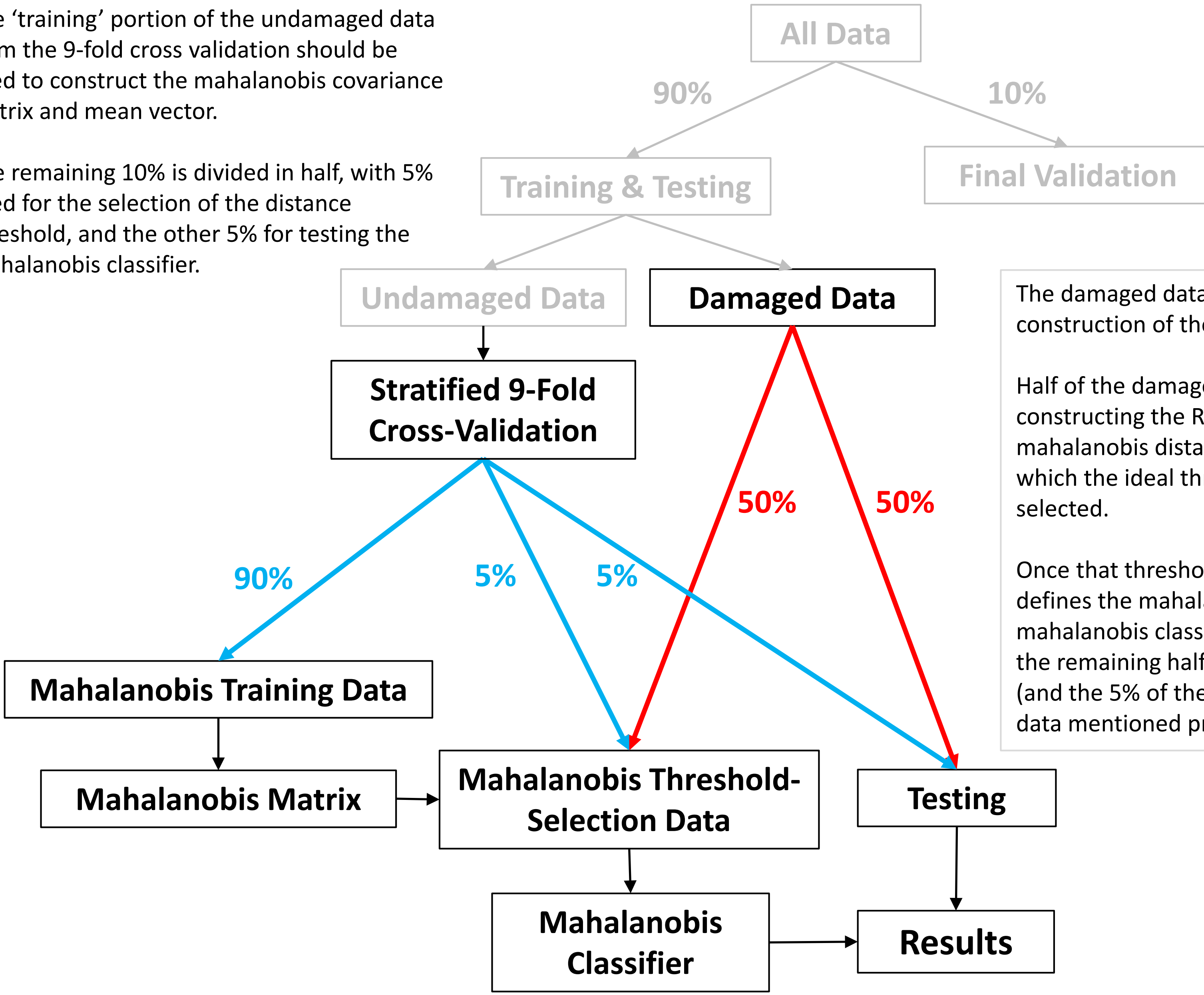
The undamaged data is then further sub-divided by 9-fold cross validation. 9 folds are used because the testing sets will comprise 1/9th of 90% (10%) which is the size of the final validation set.



Mahalanobis Classifier

The 'training' portion of the undamaged data from the 9-fold cross validation should be used to construct the mahalanobis covariance matrix and mean vector.

The remaining 10% is divided in half, with 5% used for the selection of the distance threshold, and the other 5% for testing the Mahalanobis classifier.



The damaged data is not used in the construction of the mahalanobis matrix.

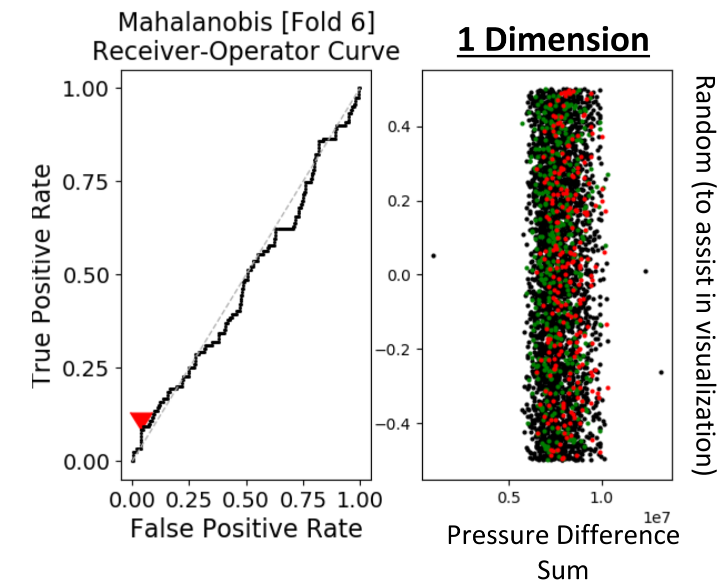
Half of the damaged data is used for constructing the ROC curve using the mahalanobis distance metric, from which the ideal threshold will be selected.

Once that threshold is selected (which defines the mahalanobis classifier), the mahalanobis classifier can be tested with the remaining half of the damaged data (and the 5% of the undamaged testing data mentioned previously).

Eload Performance with Varying # of Dimensions

The Board401 (0K Actuations) and the higher actuation datasets (28K, 50K, etc.) were used for these figures. Eload severity below 650 was considered undamaged, and equal or above 650 damaged.

1 Dimension

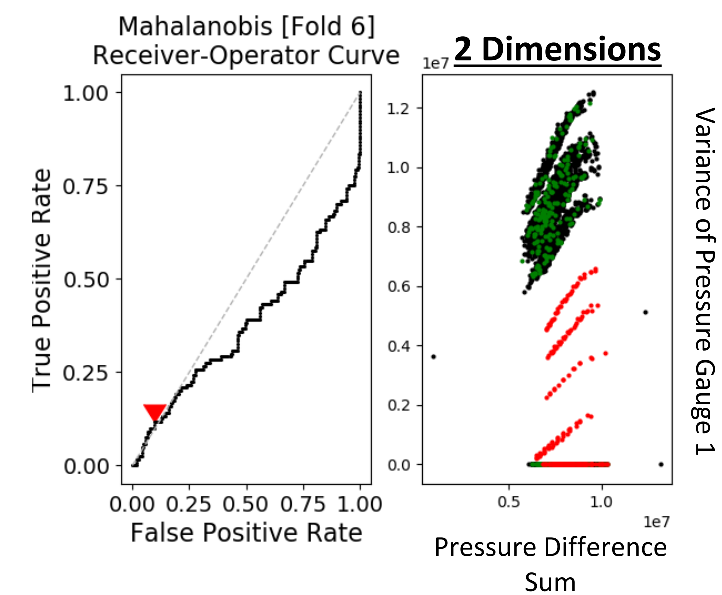


Performance using Varying # of Dimensions

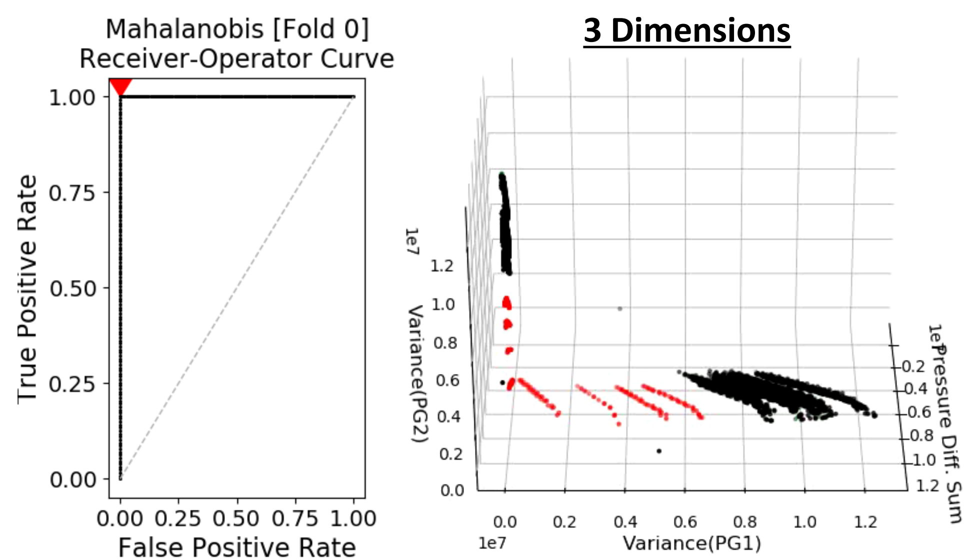
In this case, the performance becomes very good with sufficiently many dimensions. The reason for the giant leap in the jump from 2 to 3 dimensions is because the line of points (at Variance(PG1) = 0) become spread out across the Variance(PG2) Pressure Diff. Sum plane.

Theoretically, you can combine sensors to project into 2D dimensions that are about as good for separating the data, but you still physically need to capture this sensor data before doing the dimension reduction on the data.

2 Dimensions



3 Dimensions

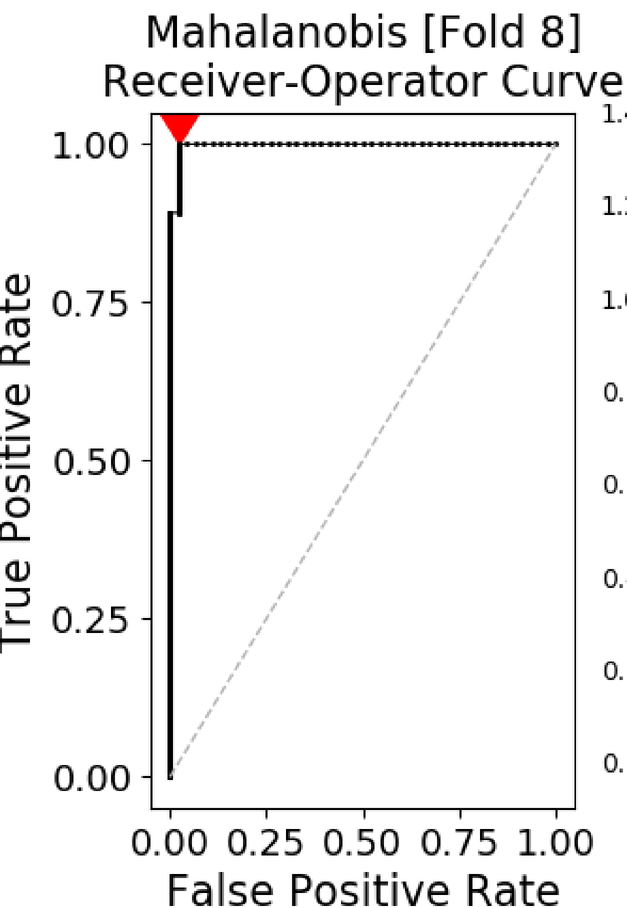


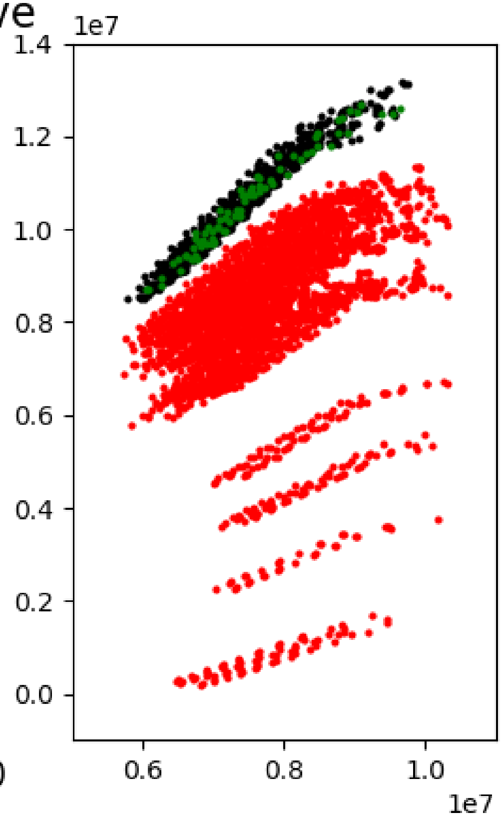
Just [000] as undamaged

ELoad

Using Board401 (0K) dataset & the higher-actuator datasets (28K, 50K, etc).

Fold 0:	Test acc: 100.000%	|	Train acc: 100.000%
Fold 1:	Test acc: 99.938%	|	Train acc: 100.000%
Fold 2:	Test acc: 100.000%	|	Train acc: 100.000%
Fold 3:	Test acc: 100.000%	|	Train acc: 100.000%
Fold 4:	Test acc: 100.000%	|	Train acc: 100.000%
Fold 5:	Test acc: 100.000%	|	Train acc: 100.000%
Fold 6:	Test acc: 99.938%	|	Train acc: 100.000%
Fold 7:	Test acc: 99.938%	|	Train acc: 99.815%
Fold 8:	Test acc: 99.877%	|	Train acc: 99.938%





Using just [000] works well perhaps because there's not much overlap between its and those of the other severity classes.

Likewise, using [000-650] works well too, because the block of intermingling points [300-650] isn't being split across the undamaged and damaged classes.

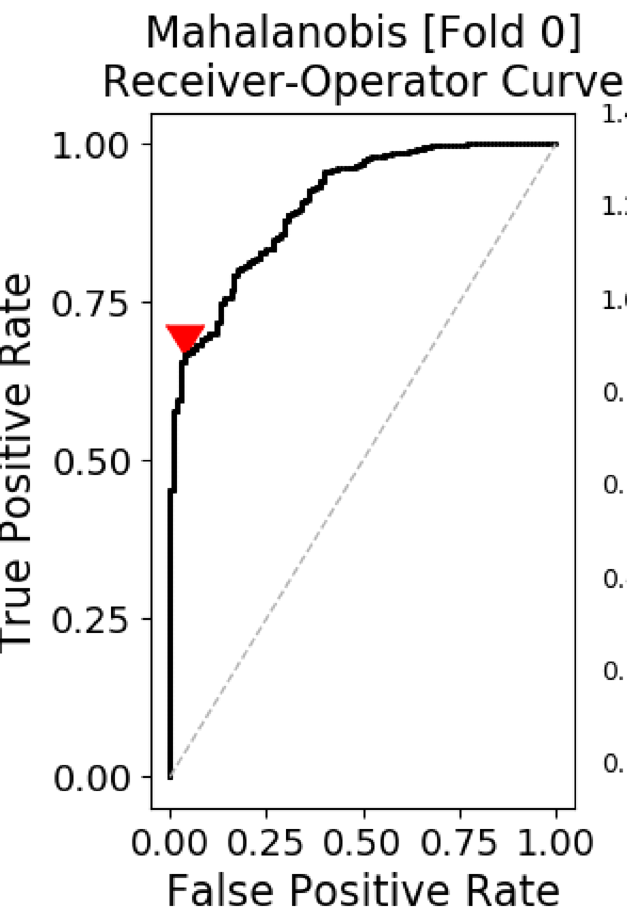
Green: Training Undamaged data
Black: Testing Undamaged data
Red: Testing Damaged data

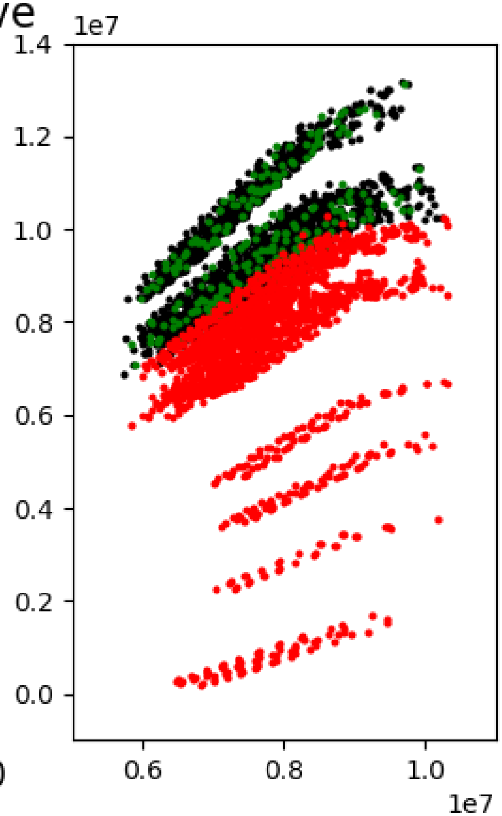
Using [000 – 375] as undamaged

ELoad

Using Board401 (0K) dataset & the higher-actuator datasets (28K, 50K, etc).

Fold 0:	Test acc: 72.866%	Train acc: 70.092%
Fold 1:	Test acc: 73.143%	Train acc: 74.161%
Fold 2:	Test acc: 79.654%	Train acc: 83.927%
Fold 3:	Test acc: 77.721%	Train acc: 77.823%
Fold 4:	Test acc: 78.433%	Train acc: 78.128%
Fold 5:	Test acc: 71.617%	Train acc: 75.178%
Fold 6:	Test acc: 81.994%	Train acc: 80.875%
Fold 7:	Test acc: 72.431%	Train acc: 75.381%
Fold 8:	Test acc: 79.145%	Train acc: 78.128%





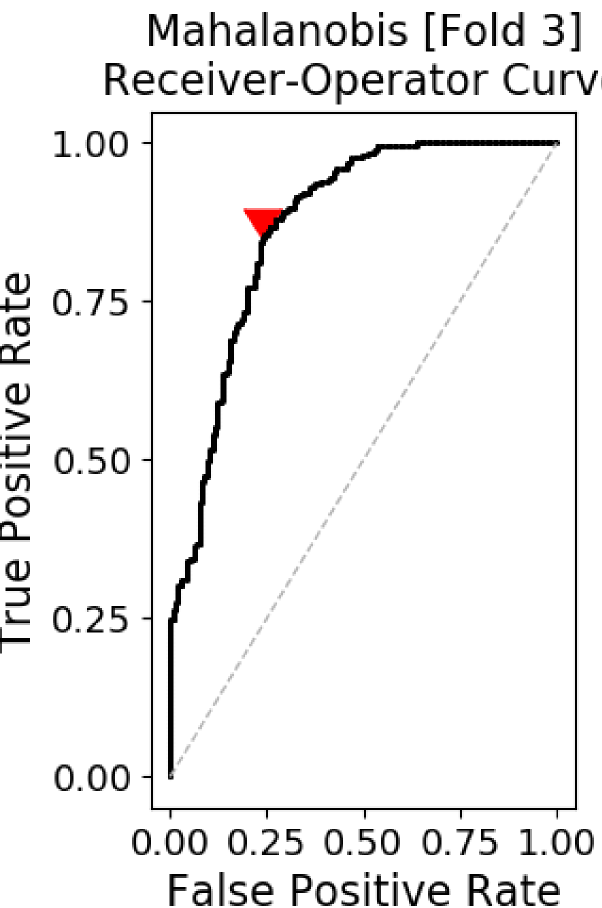
Undamaged and damaged points are intermingling because each severity in here is measured at multiple points in time after substantially different numbers of actuation, and the variation in behavior over actuation is enough to make a fuzzy border between damage severities.

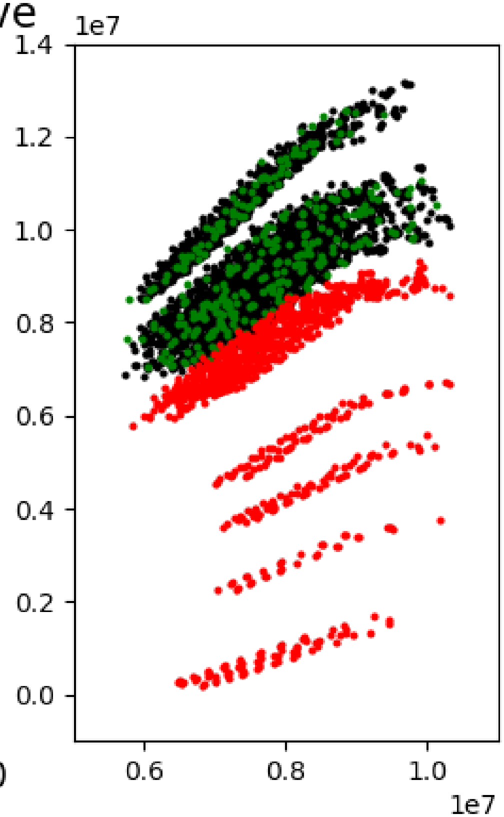
Using [000 – 480] as undamaged

ELoad

Using Board401 (0K) dataset & the higher-actuator datasets (28K, 50K, etc).

Fold 0:	Test acc: 84.746%	Train acc: 85.978%
Fold 1:	Test acc: 86.133%	Train acc: 87.827%
Fold 2:	Test acc: 88.290%	Train acc: 86.595%
Fold 3:	Test acc: 81.510%	Train acc: 83.051%
Fold 4:	Test acc: 89.368%	Train acc: 90.139%
Fold 5:	Test acc: 76.733%	Train acc: 79.199%
Fold 6:	Test acc: 88.444%	Train acc: 88.272%
Fold 7:	Test acc: 83.359%	Train acc: 89.198%
Fold 8:	Test acc: 85.670%	Train acc: 87.346%



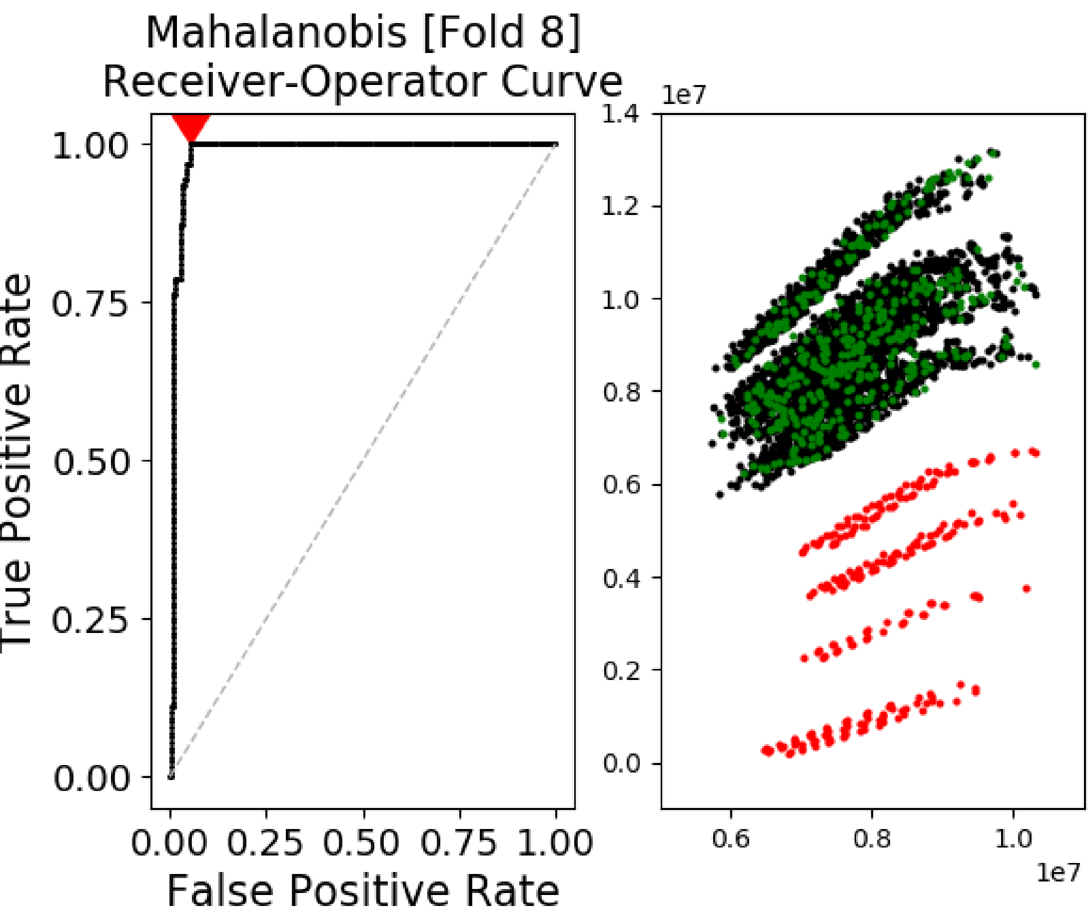


Using [000 – 650] as undamaged

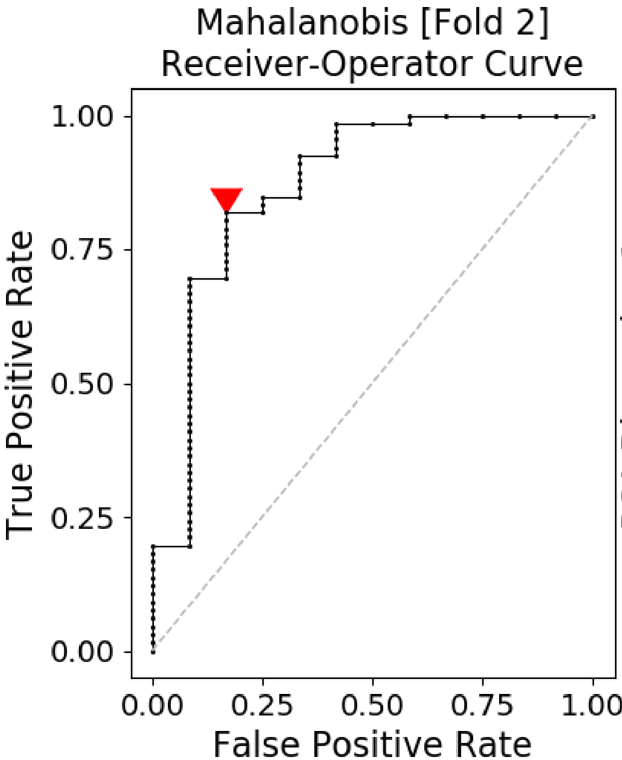
ELoad

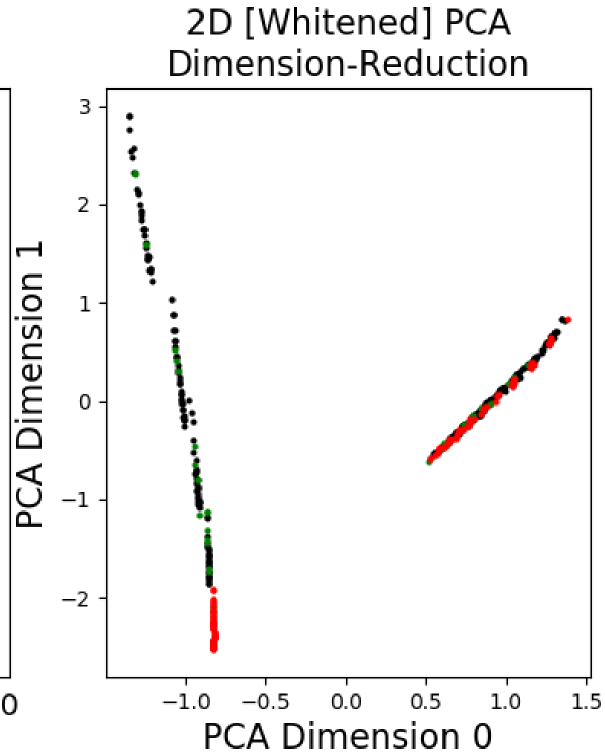
Using Board401 (0K) dataset & the higher-actuator datasets (28K, 50K, etc).

Fold 0:	Test acc: 99.381%	Train acc: 99.068%
Fold 1:	Test acc: 99.381%	Train acc: 99.379%
Fold 2:	Test acc: 97.833%	Train acc: 98.447%
Fold 3:	Test acc: 98.447%	Train acc: 99.379%
Fold 4:	Test acc: 99.068%	Train acc: 97.516%
Fold 5:	Test acc: 100.000%	Train acc: 99.689%
Fold 6:	Test acc: 98.758%	Train acc: 98.758%
Fold 7:	Test acc: 99.379%	Train acc: 99.379%
Fold 8:	Test acc: 97.205%	Train acc: 96.894%



Mahalanobis on ELeak Data





Accuracy for Each Fold [Using separate train/test for ROC threshold selection]

Fold 0:	Test acc: 85.897% |	Train acc: 96.154%
Fold 1:	Test acc: 79.487% |	Train acc: 87.179%
Fold 2:	Test acc: 73.077% |	Train acc: 82.051%
Fold 3:	Test acc: 88.462% |	Train acc: 96.104%
Fold 4:	Test acc: 75.641% |	Train acc: 83.117%
Fold 5:	Test acc: 89.744% |	Train acc: 88.312%
Fold 6:	Test acc: 89.744% |	Train acc: 92.208%
Fold 7:	Test acc: 88.462% |	Train acc: 85.714%
Fold 8:	Test acc: 67.949% |	Train acc: 72.727%

Summary

Accuracy is not great [68% - 89%] using Mahalanobis distance over the ELeak data.

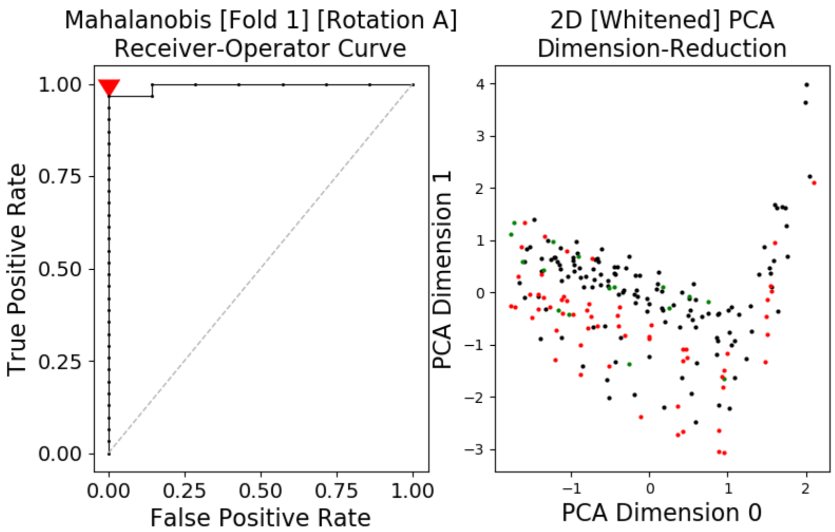
Here Eleak 005, 010, 020, and 050 were considered undamaged, and Eleak 100 & 150 damage.

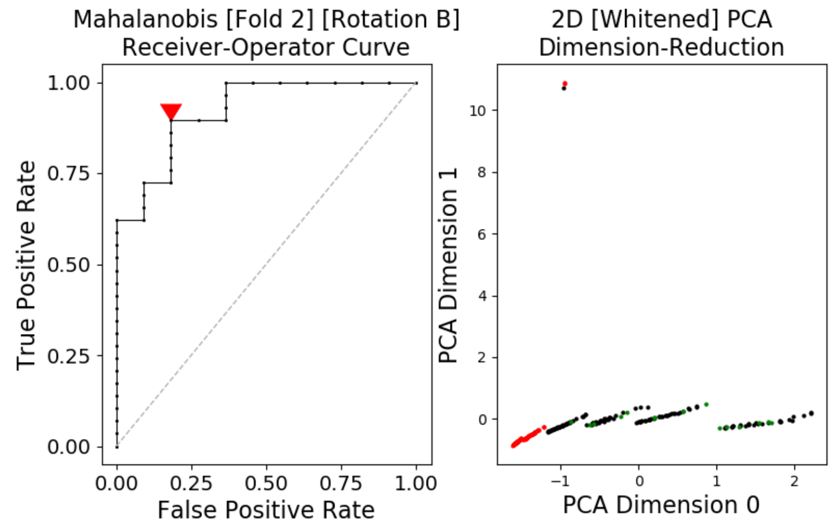
However, it looks like the larger line-cluster on the left of the 2D PCA projection scatterplot is much better separated than the line-cluster on the right.

They may need to be separated and treated differently (e.g. each given their own PCA reduction).

Mahalanobis on ELeak Data

The distribution of the points in the PCA-reduced spaces are very different – suggesting there's a definite difference in how Eleak manifests in the clockwise vs. counter-clockwise cases, besides the sign/value of the angle.





Rotation A Accuracy

[Rot. A is > median(angle)]

Fold 0:	Test acc: 94.737% |	Train acc: 94.737%
Fold 1:	Test acc: 78.947% |	Train acc: 97.368%
Fold 2:	Test acc: 86.842% |	Train acc: 89.474%
Fold 3:	Test acc: 92.105% |	Train acc: 89.474%
Fold 4:	Test acc: 76.316% |	Train acc: 81.579%
Fold 5:	Test acc: 94.737% |	Train acc: 97.368%
Fold 6:	Test acc: 94.737% |	Train acc: 100.000%
Fold 7:	Test acc: 92.105% |	Train acc: 89.474%
Fold 8:	Test acc: 89.474% |	Train acc: 89.189%

Low: 76%
High: 94%

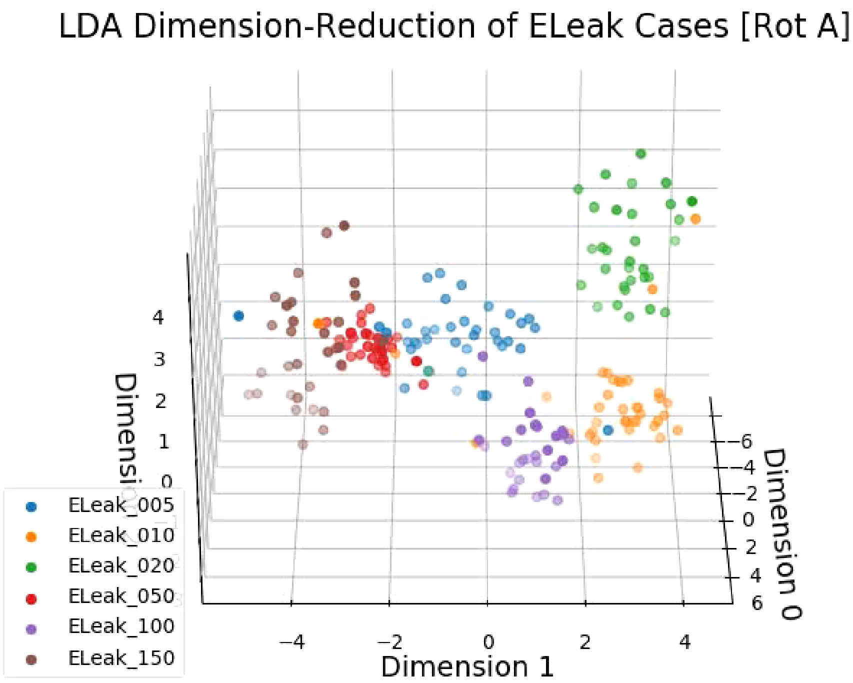
Rotation B Accuracy

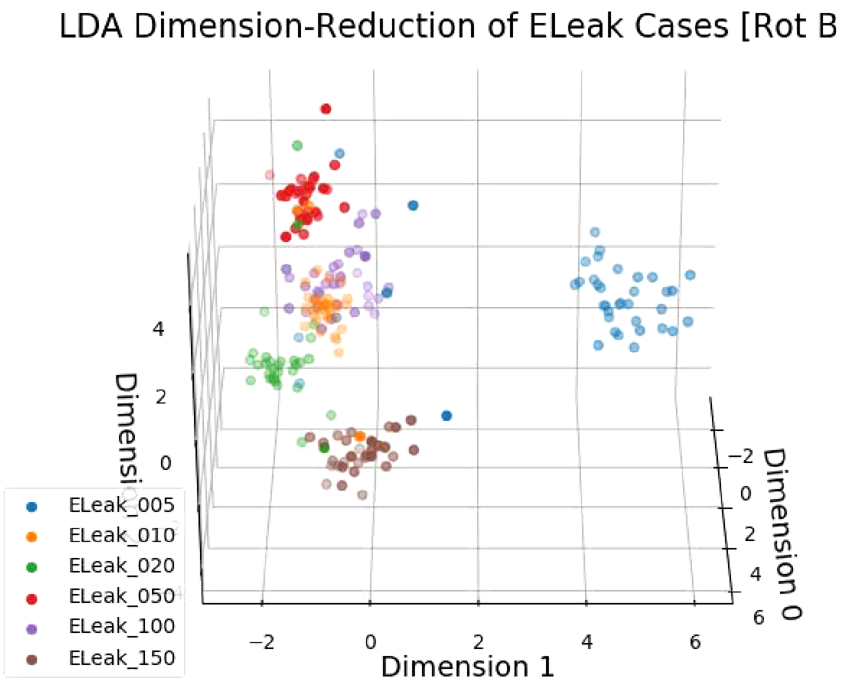
[Rot. B is <= median(angle)]

Fold 0:	Test acc: 90.000% |	Train acc: 87.500%
Fold 1:	Test acc: 82.500% |	Train acc: 85.000%
Fold 2:	Test acc: 90.000% |	Train acc: 87.500%
Fold 3:	Test acc: 95.000% |	Train acc: 90.000%
Fold 4:	Test acc: 87.500% |	Train acc: 97.436%
Fold 5:	Test acc: 97.500% |	Train acc: 92.308%
Fold 6:	Test acc: 90.000% |	Train acc: 94.872%
Fold 7:	Test acc: 90.000% |	Train acc: 94.872%
Fold 8:	Test acc: 97.500% |	Train acc: 87.179%

Low: 82%
High: 97%

LDA Dimension-Reduction on ELeak Data





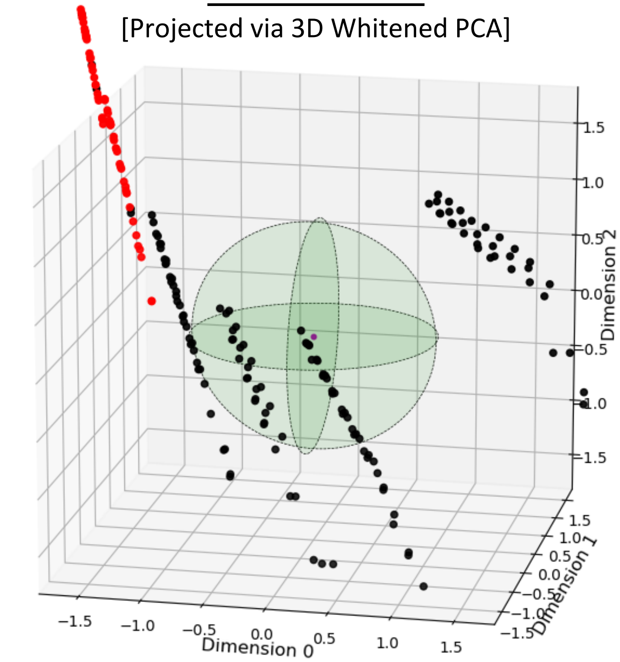
The purple and brown clusters (Eleak 100 and Eleak 150 respectively) are the damaged cases – the others are undamaged. In both 3D representations, the damaged and undamaged clusters are visually separable by a plane. This means that the classes of data are (mostly) linearly separable.

However, each LDA-found cluster still contains some outliers from clusters of another class (e.g. some blue points [Eleak 005] appear in the brown cluster [Eleak 150]).

Data Sphering

ELeak Rot. B

[Projected via 3D Whitened PCA]



Above: Rotation B ELeak data down-projected into the top 3 whitened principal component directions, with the unit sphere centered at the mean of the undamaged data. Damaged points [>=100] are in red, undamaged [<100] in black.

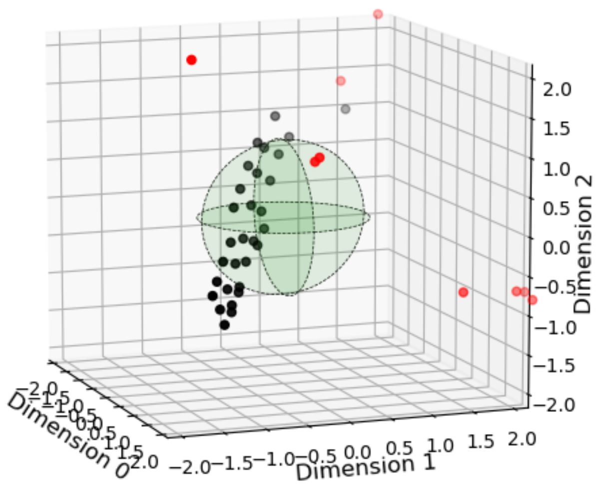
The data is not Gaussian-distributed, nor spherical after the whitening PCA transformation.

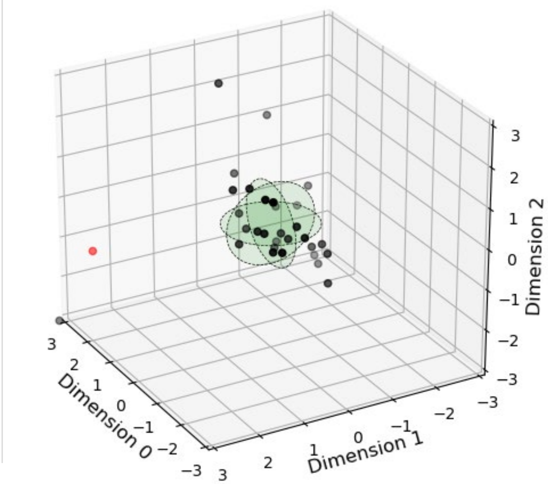
The fact that each cluster is a line [1D] in 3D space suggests the difference between the undamaged groups is overpowering the difference within the groups in 2 of the 3 dimensions.

Non-Gaussian Data

The undamaged data is not spherical after transformation.
This is because:

1. Each severity has a different mean
2. The clockwise/counter-clockwise samples have different means
3. There are strong outliers (potentially samples from previous or following severity)





Fitting only to Non-Outliers

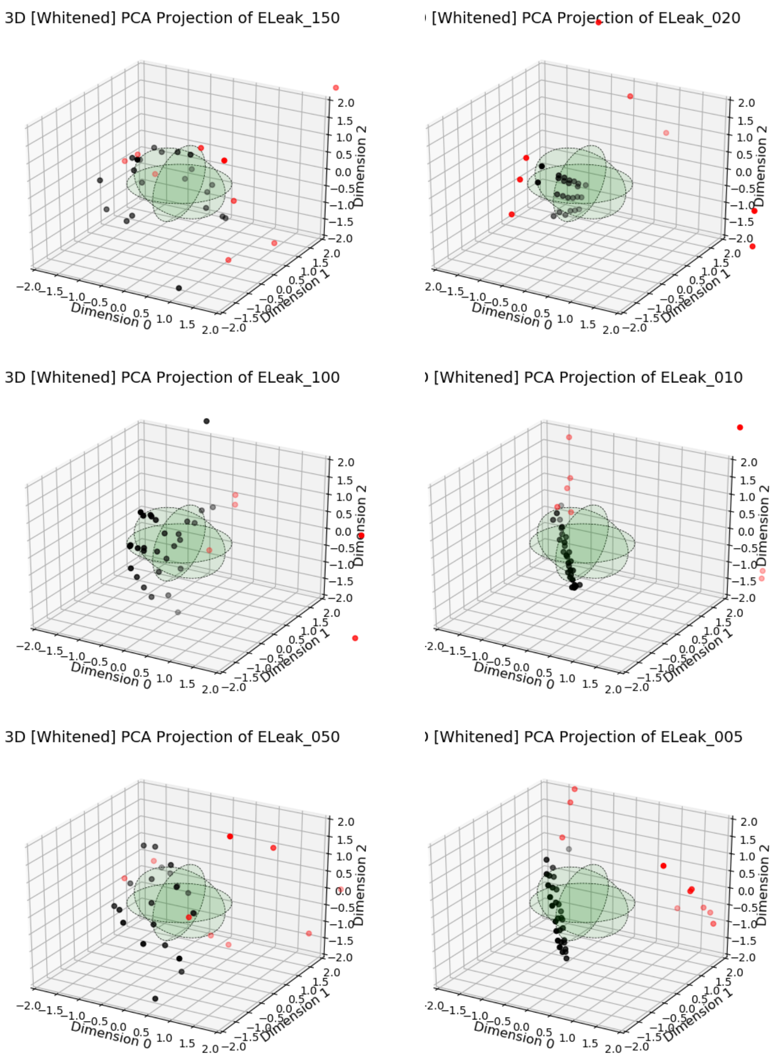
Left: Rot. B Eleak 005 data using Ali's features & down-projecting to 3D using whitened PCA.
Right: Rot. B Eleak 005 data using Ali's features & down-projecting to 3D using whitened PCA, but fitting the PCA transformation *only to non-outlier points*. An sklearn isolation forest (default settings) was used for outlier detection.

By removing non-outlier points prior to fitting the whitening PCA transformation, the resultant transformation better spheres the data.

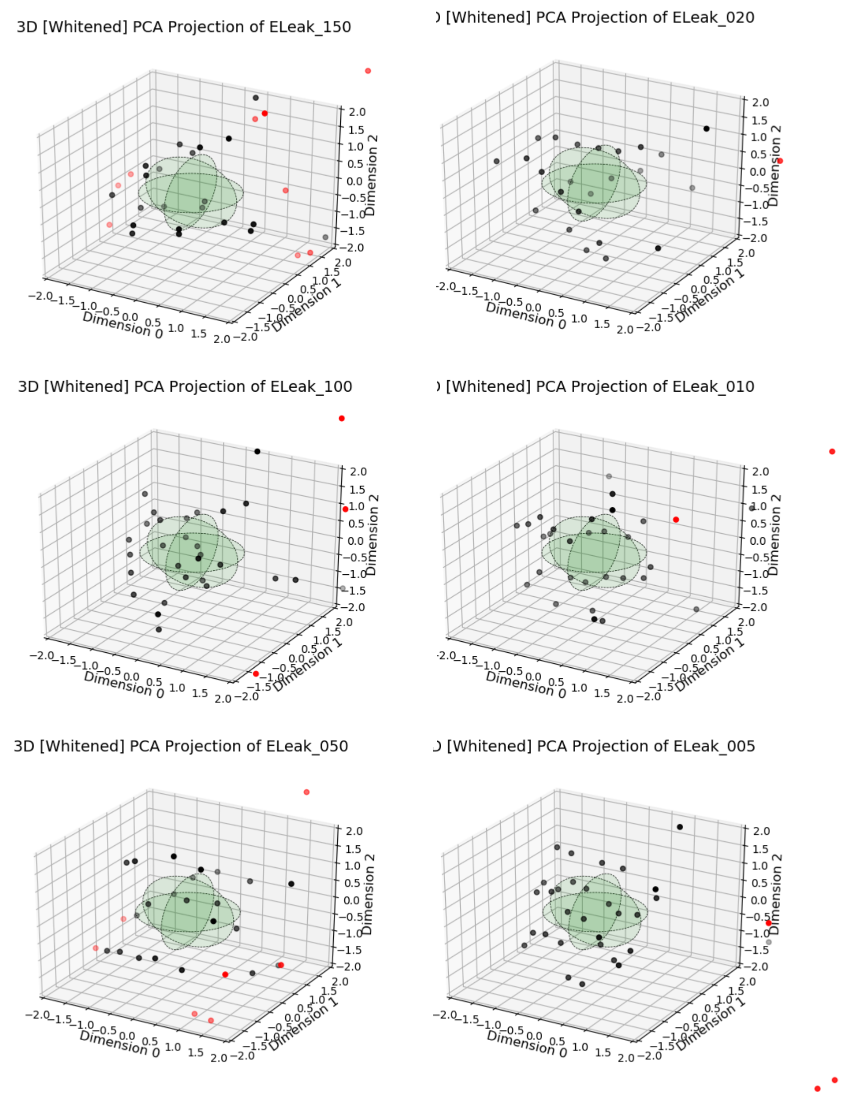
The outlier points are still plotted: a red dot can be seen on the left side of the graph. However, the other outliers are so far from the mean under the new transformation that they don't appear on the graph.

Data Sphering

Whitening PCA Projection



Whitening PCA Projection Fit only on non-outliers

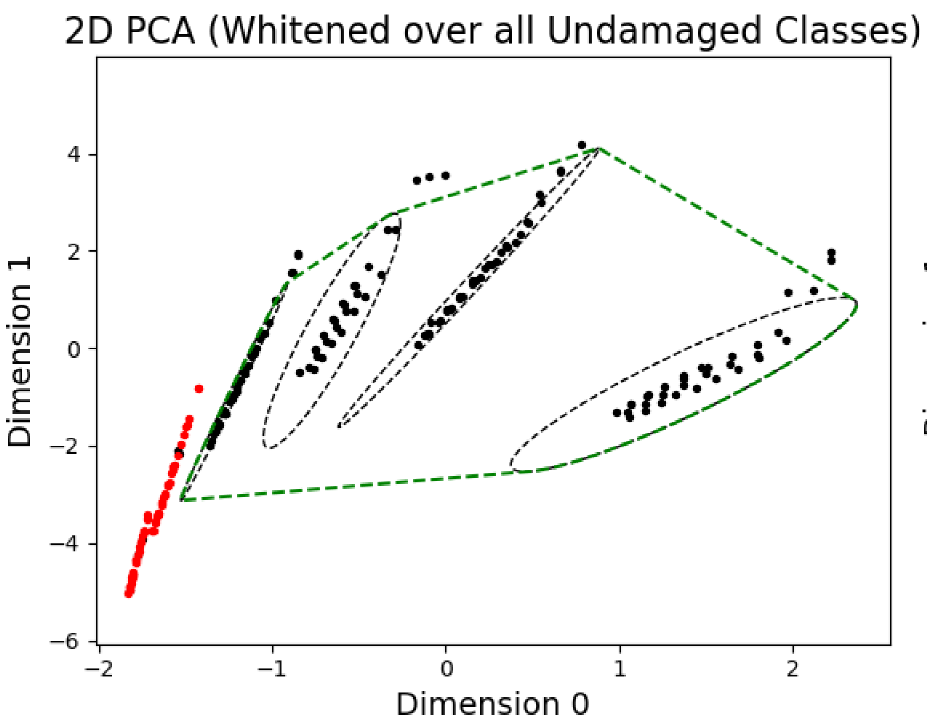


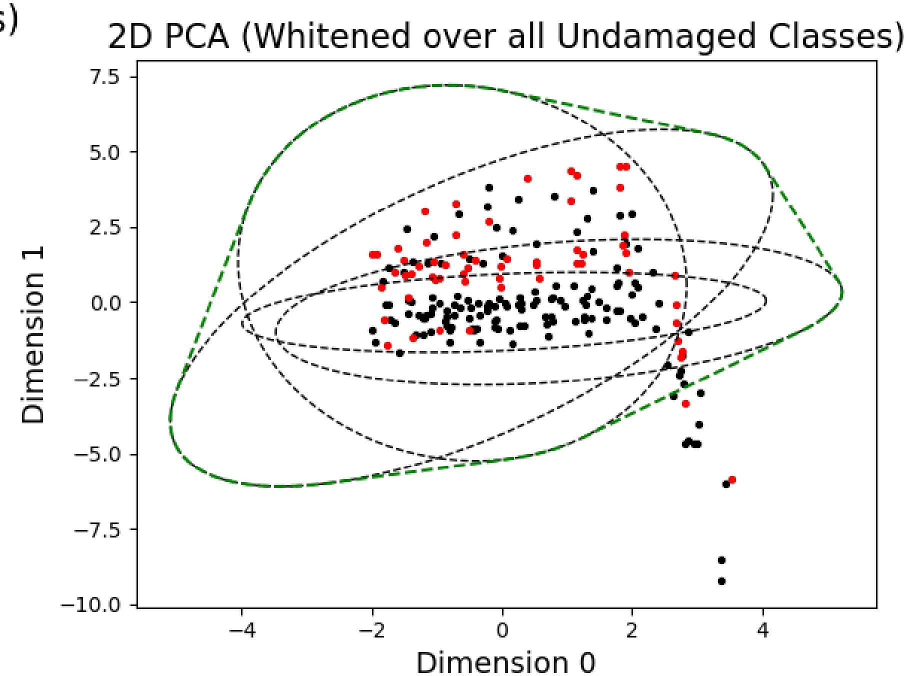
The more spread out the points are within the sphere, the better.
By contrast, a line/plane inside the sphere indicates just a few points are strongly influencing PCA’s choice of a particular dimension.

Multiple Mahalanobis Classifiers

Possible solution

Make a classifier for each severity case, then grouping classifiers according to their over-arching category (e.g. damaged/undamaged)





An ellipse is fit to each undamaged class (using isolation forest + whitening PCA), and then a convex hull is formed encapsulating all the ellipses. The size of the ellipses (i.e. the Mahalanobis distance from the center of each class) could be optimized with ROCs.

The graphics above uses a size of 5 for all the ellipses, and uses Eleak cases <100 for undamaged, and >=100 for damage. Rotation B (angle <= median) is used for the left figure, and Rotation A (angle > median) is used for the figure on the right.